



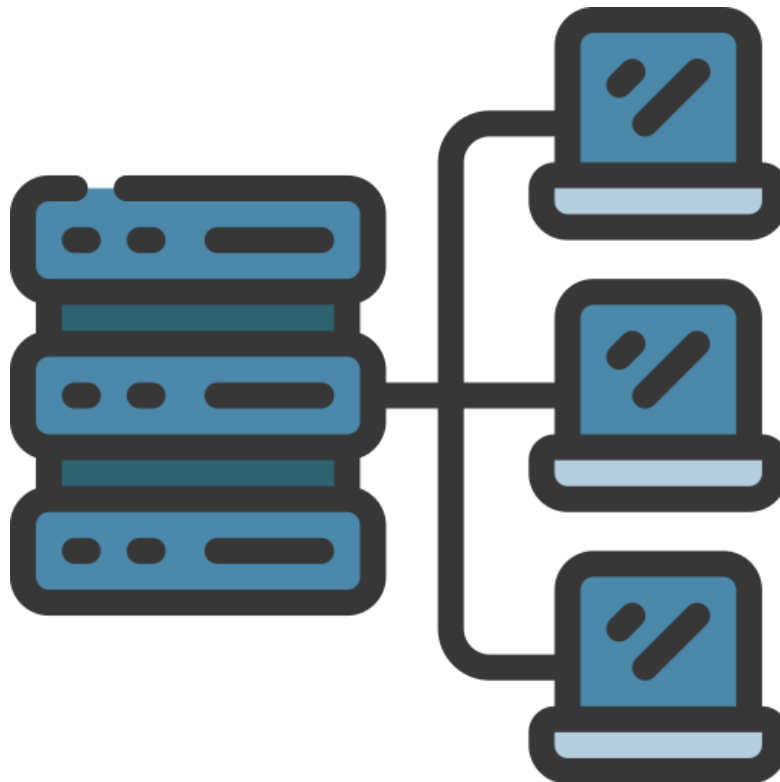
UNIVERSIDAD DE CASTILLA LA
MANCHA ESCUELA SUPERIOR
DE INFORMÁTICA

LABORATORIO 1

Seguridad en Redes
Alonso Villamayor
Moreno

[Alonso.Villamayor
@alu.uclm.es](mailto:Alonso.Villamayor@alu.uclm.es)

Curso 2023/2024



Contenidos

Seguridad en Redes.....	1
Curso 2023/2024	1
Análisis individual de Bob:	3
Análisis estático del binario Bob:	3
Análisis en ejecución del binario Bob:	5
Conclusiones Análisis individual de Bob:	6
Análisis individual de Alice:	6
Análisis estático del binario Alice:.....	7
Análisis en ejecución del binario Alice:	9
Conclusiones Análisis individual de Alice:	10
Análisis de la ejecución conjunta de ambos procesos:	10
Análisis de las llamadas al sistema:.....	10
Análisis de la red:.....	11
Análisis de la interfaz wlo1:	11
Análisis de la interfaz loopback:.....	11
DNS Falso.	12
Suplantar a Bob.	13
Conclusiones.	14

Para analizar e intentar descubrir que es lo que hacen estos dos binarios vamos a seguir los mismos pasos para ambos, primero realizaremos un análisis de los binarios por separados, primero Bob y luego Alice y observar así su comportamiento, posteriormente intentaremos hacer un seguimiento de las llamadas al sistema que realizan ambos programas por separado, y en caso de que fuese necesario realizaremos un análisis de la red en el caso de que generen tráfico o se comuniquen entre ellos.

Análisis individual de Bob:

Lo primero que hemos realizado, aunque se obvio, es ejecutar el binario Bob, no obstante, no hemos obtenido ningún tipo de salida, además el programa parece no terminar por lo que puede ser que este programa este esperando a recibir alguna información desde otro sitio.

Análisis estático del binario Bob:

Para realizar el análisis de este análisis, lo primero que hemos hecho ha sido buscar alguna herramienta para poder observar las cadenas que existan dentro del binario, ya que este puede tener información insertada dentro del propio binario o también averiguar alguna dirección de internet como una URL o algún nombre. Para ello primero hemos utilizado la herramienta objdump, hexdump, ldd y strings la cual nos permite pasar el binario a un formato hexadecimal y obtener alguna información sobre las cadenas de este binario. Para ello ejecutaremos los siguientes comandos:

- **objdump -p bob > bob_private-headers.hex:** Con la opción -p indicamos que queremos obtener los headers privados del archivo, y obtenemos información interesante y es que nos encontramos que este programa necesita de las librerías libpthread.so.0 y libc.so.6.

```
Sección Dinámica:
HASH                0x000000000050cb60
SYMTAB              0x000000000050cf60
SYMENT              0x0000000000000018
STRTAB              0x000000000050cd40
STRSZ               0x0000000000000213
RELA                0x000000000050c780
RELASZ              0x0000000000000018
RELAENT             0x0000000000000018
PLTGOT              0x0000000000579020
DEBUG               0x0000000000000000
NEEDED               libc.so.0
NEEDED               libc.so.6
VERNEED             0x000000000050cb00
VERNEEDNUM          0x0000000000000002
VERSYM              0x000000000050caa0
PLTREL              0x0000000000000007
PLTRELSZ            0x0000000000000300
JMPREL              0x000000000050c798
```

Por lo tanto, es bastante posible que este programa cree varios hilos de ejecución.

- Como hemos mencionado anteriormente es muy probable que Bob este creando varios hilos de ejecución, asique para ello vamos a probar a ejecutar el programa desde la terminal en segundo plano y comprobar el número de hilos que este tiene. Como sospechábamos Bob crea un total de 6 hilos de ejecución.

```

alonso@alonso-TUF-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1$ ./bob & ps -T | grep bob
[1] 131720
131720 131720 pts/0    00:00:00 bob
131720 131723 pts/0    00:00:00 bob
131720 131724 pts/0    00:00:00 bob
131720 131725 pts/0    00:00:00 bob
131720 131726 pts/0    00:00:00 bob
131720 131727 pts/0    00:00:00 bob

```

- **hexdump -C bob > bob.hex:** Con esta herramienta lo que conseguimos es pasar la representación del código binario a hexadecimal y además nos permite ver algunas cadenas internas del programa. No obstante, el programa parece estar cifrado de alguna manera pues la mayoría de las cadenas son ilegibles, aun así, podemos obtener alguna información extra, como que el lenguaje que se ha utilizado para construir el binario es **go** ya que encontramos algunas rutas en la maquina en la que se compilo dicho archivo, además también parece que el programa en algún momento intentará conectarse a la red, pues podemos encontrar alguna cadena dentro del binario que hace referencia a sockets. Indagando más podemos afirmar que en algún momento se conectará por red ya que podemos encontrar la llamada al sistema **syscall.sendto** y no solo esto sino que también escuchara información procedente de otro proceso o sitio web ya que también nos podemos encontrar la llamada al sistema **syscall.recvfrom**.
- Siguiendo con lo anterior y sabiendo que dicho programa es muy probable que esté utilizando sockets, he probado a filtrar la salida por tcp para ver si dicho programa utiliza tcp o udp, en principio utiliza tcp, esto lo contrastaremos más adelante cuando utilicemos alguna herramienta para analizar el tráfico de red.
- A continuación, hemos procedido a realizar un seguimiento más profundo de las dependencias del programa, para ello nos vamos a apoyar en el herramienta ldd, para ello ejecutamos el comando “**ldd bob**”, el cual no nos muestra gran cantidad de información, lo único es que encontramos la librería libpthread.so.0 cosa que ya habíamos demostrado anteriormente.

```

alonso@alonso-TUF-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1$ ldd ./bob
linux-vdso.so.1 (0x00007ffeb55fc000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f3942667000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3942475000)
/lib64/ld-linux-x86-64.so.2 (0x00007f39426a4000)

```

- Por último, vamos a realizar un análisis completo de las cadenas mediante la herramienta strings, para ello ejecutamos el siguiente comando **strings bob > Strings_bob.txt**, gracias a este comando obtenemos información sobre las diferentes librerías de go que se han necesitado para crear el binario, además también hemos obtenido rutas del ordenador sobre el que se compilo el programa como:
 - **/home/cleto/bin/go/src/internal/cpu/cpu.go**, además también hemos encontrado cierta información sobre las **estructuras** que utiliza Bob, para ello hemos filtrado la información del archivo Strings_bob.txt por la palabra struct no obstante la información no parece ser muy relevante.
 - También hemos encontrado información como que utiliza la librería mutex para sincronizar los hilos.

Análisis en ejecución del binario Bob:

Para este análisis utilizaremos las herramientas strace, la cual nos permite realizar un seguimiento de las llamadas al sistema que realiza un programa en ejecución, y también lsof para ver los archivos abiertos que maneja Bob.

- Primero vamos a ejecutar el programa con strace, para ello usaremos el siguiente comando “**strace -o llamadas_solo_bob.txt ./bob**”, como era de esperar el programa parece no terminar, no obstante, obtenemos información que nos sirve para reafirmar que Bob utiliza socket TCP como se muestra en la siguiente imagen:

```
alonsomalonso-TUF-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad en Redes_LAB1$ cat llamadas_solo_bob.txt | grep socket
socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No existe el archivo o el directorio)
socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No existe el archivo o el directorio)
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, IPPROTO_IP) = 3
```

- Además, hemos encontrado que bob intenta conectar al puerto 12345, como se muestra en la siguiente imagen:

```
alonsomalonso-TUF-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad en Redes_LAB1$ cat llamadas_solo_bob.txt | grep connect
connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No existe el archivo o el directorio)
connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No existe el archivo o el directorio)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
connect(3, {sa_family=AF_INET, sin_port=htons(12345), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operación en curso)
```

Como se puede observar en la imagen la operación se repite varias veces y se mantiene en curso, lo que puede llegar a indicar que no existe un proceso adecuado escuchando en el puerto 12345, para comprobar esto hemos utilizado el siguiente comando “**netstat -an | grep 12345**” y no hemos obtenido ninguna salida, lo que nos indica que efectivamente no existe ningún proceso escuchando en ese puerto, además en la imagen también podemos observar que la dirección IP es la 127.0.0.1 es decir nuestra propia máquina, por lo que es muy probable que Bob en cierta manera intente conectarse con un proceso local mediante la dirección 127.0.0.1:12345 dicho proceso, puede ser Alice, o algún otro proceso que el propio Alice cargue en memoria.

- Adicionalmente, también se ha encontrado, que Bob intenta realizar una operación de lectura de la dirección 127.0.0.1 por lo que parece a la espera de recibir datos.
- Respecto al número de hilos, podemos observar mediante la herramienta lsof, usando el siguiente comando “**lsof | grep bob**” como tiene una serie de archivos abiertos el proceso y además como tiene una serie de archivos por cada hilo, cabe destacar que estos archivos se repiten por hilo como muestra la siguiente imagen.

bob	307943	alonso	cwd	DIR	8,2	4096	4365056	/home/alonso/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1
bob	307943	alonso	rtd	DIR	8,2	4096	2	
bob	307943	alonso	txt	REG	8,2	1628824	4596044	/home/alonso/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1/bob
bob	307943	alonso	mem	REG	8,2	51832	6162573	/usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
bob	307943	alonso	mem	REG	8,2	2029560	6162565	/usr/lib/x86_64-linux-gnu/libc-2.31.so
bob	307943	alonso	mem	REG	8,2	157224	6162578	/usr/lib/x86_64-linux-gnu/libpthread-2.31.so
bob	307943	alonso	mem	REG	8,2	191472	6162558	/usr/lib/x86_64-linux-gnu/ld-2.31.so
bob	307943	alonso	0u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943	alonso	1u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943	alonso	2u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943	alonso	4u	a_inode	0,14	0	12533	[eventpoll]
bob	307943	alonso	5r	FIFO	0,13	0t0	668223	pipe
bob	307943	alonso	6w	FIFO	0,13	0t0	668223	pipe
bob	307943 307944 bob	alonso	cwd	DIR	8,2	4096	4365056	/home/alonso/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1
bob	307943 307944 bob	alonso	rtd	DIR	8,2	4096	2	
bob	307943 307944 bob	alonso	txt	REG	8,2	1628824	4596044	/home/alonso/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1/bob
bob	307943 307944 bob	alonso	mem	REG	8,2	51832	6162573	/usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
bob	307943 307944 bob	alonso	mem	REG	8,2	2029560	6162565	/usr/lib/x86_64-linux-gnu/libc-2.31.so
bob	307943 307944 bob	alonso	mem	REG	8,2	157224	6162578	/usr/lib/x86_64-linux-gnu/libpthread-2.31.so
bob	307943 307944 bob	alonso	mem	REG	8,2	191472	6162558	/usr/lib/x86_64-linux-gnu/ld-2.31.so
bob	307943 307944 bob	alonso	0u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943 307944 bob	alonso	1u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943 307944 bob	alonso	2u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943 307944 bob	alonso	4u	a_inode	0,14	0	12533	[eventpoll]
bob	307943 307944 bob	alonso	5r	FIFO	0,13	0t0	668223	pipe
bob	307943 307944 bob	alonso	6w	FIFO	0,13	0t0	668223	pipe
bob	307943 307945 bob	alonso	cwd	DIR	8,2	4096	4365056	/home/alonso/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1
bob	307943 307945 bob	alonso	rtd	DIR	8,2	4096	2	
bob	307943 307945 bob	alonso	txt	REG	8,2	1628824	4596044	/home/alonso/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad_en_Red_LAB1/bob
bob	307943 307945 bob	alonso	mem	REG	8,2	51832	6162573	/usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
bob	307943 307945 bob	alonso	mem	REG	8,2	2029560	6162565	/usr/lib/x86_64-linux-gnu/libc-2.31.so
bob	307943 307945 bob	alonso	mem	REG	8,2	157224	6162578	/usr/lib/x86_64-linux-gnu/libpthread-2.31.so
bob	307943 307945 bob	alonso	mem	REG	8,2	191472	6162558	/usr/lib/x86_64-linux-gnu/ld-2.31.so
bob	307943 307945 bob	alonso	0u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943 307945 bob	alonso	1u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943 307945 bob	alonso	2u	CHR	136,0	0t0	3	/dev/pts/0
bob	307943 307945 bob	alonso	4u	a_inode	0,14	0	12533	[eventpoll]
bob	307943 307945 bob	alonso	5r	FIFO	0,13	0t0	668223	pipe
bob	307943 307945 bob	alonso	6w	FIFO	0,13	0t0	668223	pipe

Cabe destacar que esta estructura se repite hasta la Id del hilo 307948, no obstante, estas ids entre ejecución y ejecución pueden variar.

Conclusiones Análisis individual de Bob:

- Bob crea varios hilos probablemente con el objetivo de que alguno de ellos se encargue de manejar la información que se le pase por red y el reste pueden realizar tareas como encriptar mensajes o procesar los mensajes para realizar alguna acción en el ordenador del usuario.
- Tras ejecutar Bob, hemos comprobado que el propietario de los diferentes procesos es el mismo que con el que se lanza, por lo que Bob no parece intentar escalar de alguna manera en privilegios dentro del sistema.
- A nivel interno no parece realizar nada, pero intenta conectar a la dirección 127.0.0.1:12345 por lo que es muy probable que al analizarlo junto a Alice obtengamos otros resultados

Con esta información, vamos a probar a poner un proceso de netcat a la escucha en el puerto 12345 y ejecutar Bob a ver que sucede, para ello ejecutamos el siguiente comando: “**nc -l -p 12345**”, con esto conseguimos que Bob hable con nuestro servidor, el problema que ocurre es que al intentar realizar el **handshake** entre las dos partes, obtenemos fallos, probablemente porque Bob este demandando algún tipo de clave para cifrar la conexión. Esto se muestra en la siguiente imagen:

14	6.297934398	127.0.0.1	127.0.0.1	TCP	76	41226 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM.
15	6.297948935	127.0.0.1	127.0.0.1	TCP	56	12345 → 41226 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
16	7.299063465	127.0.0.1	127.0.0.1	TCP	76	41232 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM.
17	7.299106024	127.0.0.1	127.0.0.1	TCP	56	12345 → 41232 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
18	8.300402775	127.0.0.1	127.0.0.1	TCP	76	41234 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM.
19	8.300415851	127.0.0.1	127.0.0.1	TCP	56	12345 → 41234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
20	9.301005692	127.0.0.1	127.0.0.1	TCP	76	41248 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM.
21	9.301038657	127.0.0.1	127.0.0.1	TCP	56	12345 → 41248 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
22	10.301783855	127.0.0.1	127.0.0.1	TCP	76	41250 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM.
23	10.301816298	127.0.0.1	127.0.0.1	TCP	56	12345 → 41250 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
24	11.302805257	127.0.0.1	127.0.0.1	TCP	76	41254 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM.
25	11.302838698	127.0.0.1	127.0.0.1	TCP	56	12345 → 41254 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Por lo tanto, es probable que para averiguar más funcionalidades sobre el programa Bob sea necesario obtener más información sobre Alice, o ejecutarlos conjuntamente.

Análisis individual de Alice:

Para analizar la funcionalidad de Alice, vamos a utilizar las mismas herramientas que hemos usado con Bob.

Análisis estático del binario Alice:

- **objdump -p alice > alice_private_heders.txt:** En este caso la información que obtenemos es muy parecida a la que hemos obtenido con Bob, ya que encontramos que necesita las librerías `libpthread.so.0` y `libc.so.6` de tal modo que al igual que Bob es muy probable que Alice cree varios hilos de ejecución. Para comprobar si Alice ejecuta más de un hilo, vamos a lanzarlo en segundo plano en una terminal, y posteriormente obtendremos el número de hilos.

```
alonso@alonso-TUP-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad en Redes_LAB1$ ps -T |
grep alice
179210 179210 pts/0    00:00:00 alice
179210 179211 pts/0    00:00:00 alice
179210 179212 pts/0    00:00:00 alice
179210 179213 pts/0    00:00:00 alice
179210 179214 pts/0    00:00:00 alice
179210 179215 pts/0    00:00:00 alice
179210 179216 pts/0    00:00:00 alice
179210 179217 pts/0    00:00:00 alice
179210 179218 pts/0    00:00:00 alice
179210 179219 pts/0    00:00:00 alice
179210 179220 pts/0    00:00:00 alice
179210 179221 pts/0    00:00:00 alice
alonso@alonso-TUP-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad en Redes_LAB1$ ps -T |
grep alice | wc -l
12
alonso@alonso-TUP-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad en Redes_LAB1$
```

En este caso, podemos observar que Alice crea un total de 8 hilos.

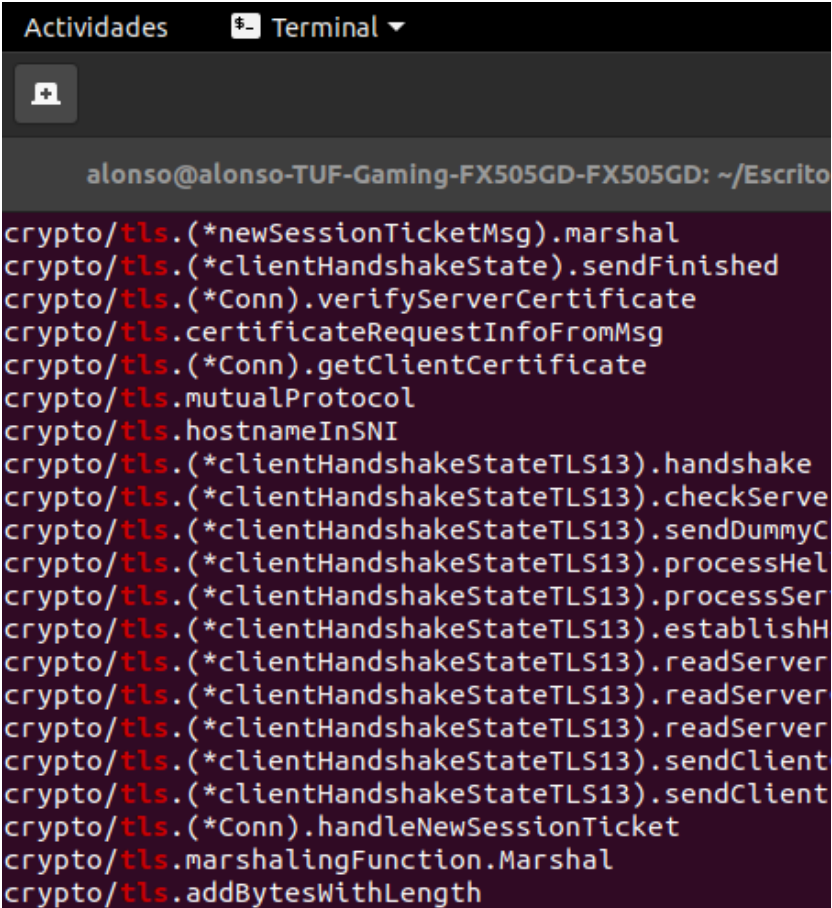
- **hexdump -C alice > alice.hex:** A continuación, hemos probado a pasar el binario Alice a hexadecimal con el objetivo de buscar alguna cadena de texto que nos pueda proporcionar información. Una vez realizado esto hemos encontrado algunas llamadas como **syscall.socket** por lo que al igual que en el caso de Bob, Alice, muy posiblemente está utilizando socket para acceder a la red o comunicarse con otros procesos. También y suponiendo que Alice puede hacer de cliente hemos filtrado por la palabra **client** y hemos encontrado bastante información, como se muestra en la imagen.

```
alonso@alonso-TUP-Gaming-FX505GD-FX505GD:~/Escritorio/ESI/Cuarto/Seguridad en Redes/Seguridad en Redes_LAB1/info_alice$ cat alice.hex | grep client
001ff6d0 74 48 be 63 6c 69 65 6e 74 20 66 48 8b bc 24 e0 [th.client fin..$.]
0026e100 00 00 63 6c 69 65 6e 74 00 00 06 63 6c 6f 73 65 [..client...close]
00270a00 74 4d 44 35 00 00 09 63 6c 69 65 6e 74 53 75 6d [TMD5...clientSum]
00273d30 63 6c 69 65 6e 74 52 61 6e 64 6f 6d 00 00 0c 63 [clientRandom...c]
00275e90 63 6c 69 65 6e 74 50 72 6f 74 6f 63 6f 6c 00 00 [clientProtocol...]
00279ee0 72 2e 63 6c 69 65 6e 74 40 65 6c 6e 6f 4d 73 67 [ls.clientHelloMsg]
0027a0f0 70 32 63 6c 69 65 6e 74 43 6f 6e 50 6f 6f 6c [p2clientConnPool]
0027b5b0 00 00 15 63 6c 69 65 6e 74 46 69 6e 69 73 68 65 [...clientFinishe]
0027c320 74 70 2e 68 74 74 70 32 63 6c 69 65 6e 74 53 74 [tp.http2clientSt]
0027c800 68 74 74 70 2e 68 74 74 70 32 63 6c 69 65 6e 74 [http.http2client]
0027d240 70 32 63 6c 69 65 6e 74 43 6f 6e 50 6f 6f 6c [p2clientConnPool]
0027d480 00 00 19 2a 74 6c 73 2e 63 6c 69 65 6e 74 4b 65 [...*tls.clientKe]
0027da50 63 6c 69 65 6e 74 53 74 72 65 61 6d 00 00 1b 2a [clientStream...*]
0027ea80 63 6c 69 65 6e 74 48 61 6e 64 73 68 61 6b 65 53 [clientHandshakeS]
0027f020 65 2e 63 6c 69 65 6e 74 45 76 65 6e 74 43 6f 6e [e.clientEventCon]
0027f9e0 70 32 63 6c 69 65 6e 74 53 74 72 65 61 6d 00 00 [p2clientStream..]
002804f0 74 74 70 32 63 6c 69 65 6e 74 53 74 72 65 61 6d [http2clientStream]
00280670 70 32 63 6c 69 65 6e 74 53 74 72 65 61 6d 00 00 [p2clientStream..]
00280bb0 2e 68 74 74 70 32 63 6c 69 65 6e 74 53 74 72 65 [http2clientStre]
002810b0 72 3b 20 6d 20 2a 74 6c 73 2e 63 6c 69 65 6e 74 [r; m *tls.client]
00297e00 63 6c 69 65 6e 74 43 6f 6e 6e 50 6f 6f 6c 3b 20 [clientConnPool; ]
002980e0 2a 74 6c 73 2e 63 6c 69 65 6e 74 4b 65 79 45 78 [*tls.clientKeyEx]
002a4400 6c 73 2e 63 6c 69 65 6e 74 48 65 6c 6c 6f 4d 73 [ls.clientHelloMs]
002a4430 2a 74 6c 73 2e 63 6c 69 65 6e 74 4b 65 79 45 78 [*tls.clientKeyEx]
002a57d0 2e 63 6c 69 65 6e 74 48 65 6c 6c 6f 4d 73 67 2c [..clientHelloMsg,]
002ba2b0 74 74 70 32 63 6c 69 65 6e 74 53 74 72 65 61 6d [tp2clientStream]
002ce000 74 74 70 2d 63 6c 69 65 6e 74 2f 31 2e 31 47 6f [http-client/1.1Go]
002ece90 2d 68 74 74 70 2d 63 6c 69 65 6e 74 2f 32 2e 30 [-http-client/2.0]
```

Parece que existe un **helloMsg** que Alice enviará a algún sitio, también podemos observar cómo aparece **http2client** por lo que puede ser que Alice realice algún tipo de petición http. También se puede observar que aparece varias veces repetida la palabra **tls**, por lo que es muy posible que Alice utilice este protocolo para comunicarse y cifrar los mensajes, además se observa la palabra **dns** por lo que puede ser que Alice intente conectarse a un servidor externo. Por último, también hemos probado a buscar palabras clave como **url**, **send**, **sendto**, **tls**, **key**, y lo que hemos obtenido es que, al igual que Bob, el programa está hecho con **go**, utiliza **tls** y también junto a **tls** aparecía la palabra **key**, por lo que es muy probable que necesitemos alguna clave para descifrar los mensajes,

además Alice también realiza la llamada **syscall.recvfrom** por lo que aparte de enviar información también recibe.

- **ldd alice:** A continuación, se han examinado las dependencias del binario por si existiese alguna diferente a las que tiene Bob, aunque las librerías que hemos obtenido en este caso son las mismas que las de Bob.
- **strings alice:** Por último, vamos a probar a buscar strings más profundamente en el binario, los resultados se han guardado en el archivo strings_alice.txt, apoyándonos en la información anterior, hemos probado a filtrar por la palabra **message**, y se encuentra casi siempre acompañada por la palabra **dns**, por lo que estamos prácticamente seguros de que Alice realizará alguna petición **dns**. Al filtrar por la palabra **tls** obtenemos bastante información, ya que parece que se realizan operaciones de **handshake** e intercambia algún tipo de certificado como se puede ver en la imagen.



```
Actividades Terminal ▾
alonso@alonso-TUF-Gaming-FX505GD-FX505GD: ~/Escrito
crypto/tls.(*newSessionTicketMsg).marshal
crypto/tls.(*clientHandshakeState).sendFinished
crypto/tls.(*Conn).verifyServerCertificate
crypto/tls.certificateRequestInfoFromMsg
crypto/tls.(*Conn).getClientCertificate
crypto/tls.mutualProtocol
crypto/tls.hostnameInSNI
crypto/tls.(*clientHandshakeStateTLS13).handshake
crypto/tls.(*clientHandshakeStateTLS13).checkServe
crypto/tls.(*clientHandshakeStateTLS13).sendDummyC
crypto/tls.(*clientHandshakeStateTLS13).processHel
crypto/tls.(*clientHandshakeStateTLS13).processSer
crypto/tls.(*clientHandshakeStateTLS13).establishH
crypto/tls.(*clientHandshakeStateTLS13).readServer
crypto/tls.(*clientHandshakeStateTLS13).readServer
crypto/tls.(*clientHandshakeStateTLS13).sendClient
crypto/tls.(*clientHandshakeStateTLS13).sendClient
crypto/tls.(*Conn).handleNewSessionTicket
crypto/tls.marshalingFunction.Marshal
crypto/tls.addBytesWithLength
```

En la segunda imagen, también podemos observar cómo se siguen una serie de pasos con el módulo de **tls**, además supongo que las repetidas operaciones de **marshal** y **unmarshal** se deben a que estos mensajes se están utilizando para una comunicación a través de red o a través de la maquina local, también podemos observar algunas cadenas como: **clientHelloMsg**, **serverHelloMsg**, **helloRequestMsg** por lo que estos mensajes se pueden estar utilizando en la comunicación, además podemos ver que a estos punteros, posteriormente se les aplica unas funciones de **marshaling** como las siguientes:


```

crypto/tls.(*clientHelloMsg).marshal.func1.4.12.1
crypto/tls.(*clientHelloMsg).marshal.func1.4.12
crypto/tls.(*clientHelloMsg).marshal.func1.4.13.1
crypto/tls.(*clientHelloMsg).marshal.func1.4.13
crypto/tls.(*clientHelloMsg).marshal.func1.4.14.1.1
crypto/tls.(*clientHelloMsg).marshal.func1.4.14.1
crypto/tls.(*clientHelloMsg).marshal.func1.4.14.2.1
crypto/tls.(*clientHelloMsg).marshal.func1.4.14.2
crypto/tls.(*clientHelloMsg).marshal.func1.4.14
crypto/tls.(*clientHelloMsg).marshal.func1.4
crypto/tls.(*clientHelloMsg).marshal.func1
crypto/tls.(*clientHelloMsg).updateBinders.func1.1
crypto/tls.(*clientHelloMsg).updateBinders.func1
crypto/tls.(*serverHelloMsg).marshal.func1.1
crypto/tls.(*serverHelloMsg).marshal.func1.2.1.1
crypto/tls.(*serverHelloMsg).marshal.func1.2.1
crypto/tls.(*serverHelloMsg).marshal.func1.2.2.1.1
crypto/tls.(*serverHelloMsg).marshal.func1.2.2.1

```

También hemos filtrado **por http, https y connect**, no obstante no se ha encontrado mucha información, lo único es que aparece gran cantidad de veces la palabra http en comparación con Bob por lo que probablemente Alice haga un uso mucho más exhaustivo de la red.

Análisis en ejecución del binario Alice:

- Primero vamos a comprobar las llamadas al sistema que realiza el programa, para ello usamos la herramienta **strace**, los resultados, se guardan en el archivo llamadas_solo_alice.txt, al igual que en el caso de Bob el programa parece no terminar, esto puede ser debido a que se necesiten ejecutar los dos a la vez para que ambos terminen. Al filtrar por socket, nos encontramos que, a diferencia de Bob, Alice, crea tanto un socket TCP como un socket UDP (SOCK_STREAM y SOCK_DGRAM), esto tiene mucho sentido teniendo en cuenta que en el análisis estático del programa nos hemos encontrado que este realiza una petición DNS, ya que estas peticiones se realizan mediante UDP, es decir mediante un SOCK_DGRAM. Al filtrar por **recv** nos encontramos que Alice realiza un **recv** y encontramos la cadena gitlab.com por lo que podemos pensar que la petición DNS se realiza para resolver el nombre gitlab.com. Al filtrar por la palabra **read**, nos encontramos información interesante, pues parece que Alice lee de alguna parte información de un certificado, probablemente un certificado **ssl**.
- Teniendo en cuenta la información obtenida de Bob, vamos a usar **lsof -i** para comprobar si Alice utiliza el puerto 12345, que es con el que Bob intentaba conectar, como se puede observar en la imagen:

```

alice 599210 alonso 6u IPv4 1167401 0t0 TCP alonso-TUF-Gaming-FX505GD-FX505GD:50606->172.65.251.78:https (ESTABLISHED)
alice 599210 alonso 7u IPv4 1158522 0t0 TCP localhost:12345 (LISTEN)

```

Alice, se encuentra escuchando en el puerto **12345** que es el puerto al que Bob intentaba conectarse, pero, además, también podemos ver como Alice tiene una conexión establecida con la **ip 172.65.251.78** a través del puerto **50606** no obstante se ha comprobado que este puerto varía de ejecución en ejecución. En principio la ip 172.65.251.78 se debería corresponder con gitlab.com ya que era sobre este nombre

sobre el que Alice realizaba una petición DNS, para comprobarlo se ha utilizado el comando nslookup, y como era de esperar la **ip** anteriormente mencionada coincide con el nombre gitlab.com

- Por último, vamos, vamos a utilizar **wireshark** para analizar la comunicación que Alice tiene con el exterior, como se puede apreciar en la imagen.

No.	Time	Source	Destination	Protocol	Length	Info
9	0.039516567	192.168.207.32	172.65.251.78	TCP	76	53410 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 ..
10	0.050382727	172.65.251.78	192.168.207.32	TCP	76	443 → 53410 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1400 S..
11	0.050463641	192.168.207.32	172.65.251.78	TCP	68	53410 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=297693616..
12	0.051436263	192.168.207.32	172.65.251.78	TLSv1.3	348	Client Hello
13	0.061695194	172.65.251.78	192.168.207.32	TCP	68	443 → 53410 [ACK] Seq=1 Ack=281 Win=57344 Len=0 TSval=2783728..
14	0.065109310	172.65.251.78	192.168.207.32	TLSv1.3	2116	Server Hello, Change Cipher Spec
15	0.065109572	172.65.251.78	192.168.207.32	TLSv1.3	1007	Application Data
16	0.065176410	192.168.207.32	172.65.251.78	TCP	68	53410 → 443 [ACK] Seq=281 Ack=2049 Win=62208 Len=0 TSval=2976..
17	0.065221196	192.168.207.32	172.65.251.78	TCP	68	53410 → 443 [ACK] Seq=281 Ack=2988 Win=61312 Len=0 TSval=2976..
18	0.129635610	192.168.207.32	172.65.251.78	TLSv1.3	132	Change Cipher Spec, Application Data
19	0.129720310	192.168.207.32	172.65.251.78	TLSv1.3	154	Application Data
20	0.129998306	192.168.207.32	172.65.251.78	TLSv1.3	164	Application Data
21	0.139987978	172.65.251.78	192.168.207.32	TCP	68	443 → 53410 [ACK] Seq=2988 Ack=527 Win=65536 Len=0 TSval=2783..
22	0.139988110	172.65.251.78	192.168.207.32	TLSv1.3	139	Application Data
23	0.140025128	192.168.207.32	172.65.251.78	TCP	68	53410 → 443 [ACK] Seq=527 Ack=3059 Win=64128 Len=0 TSval=2976..
24	0.140172959	192.168.207.32	172.65.251.78	TLSv1.3	99	Application Data
25	0.191172632	172.65.251.78	192.168.207.32	TCP	68	443 → 53410 [ACK] Seq=3059 Ack=558 Win=65536 Len=0 TSval=2783..

Como hemos mencionado anteriormente, Alice realiza una petición DNS, una vez resuelta, se conecta a gitlab.com y hacen el **handshake** una vez realizado esto, Alice envía por TLS el mensaje que dice **Client Hello**, mensaje que ya nos habíamos encontrado dentro de Alice cuando hemos analizado los strings que contenía su Código, además el servidor le responde **Server Hello, Change Cipher Spec** que puede ser que tenga algo que ver con el **serverHelloMsg** encontrado durante el análisis de los strings de Alice, por ultimo Alice responde con un mensaje así **Change Cipher Spec, Application Data**, por lo que esto podría ser una manera de ponerse de acuerdo en cómo encriptar las comunicaciones. Importante resaltar que Alice establece la comunicación siempre con el puerto 443. Cabe destacar que la captura de wireshark se encuentra en la carpeta info_alice y se llama captura_alice.pcapng

Conclusiones Análisis individual de Alice:

- Alice realiza una petición DNS a gitlab.com con ip 172.65.251.78, además Alice accede mediante http a un recurso de gitlab.com.
- Alice utiliza la red de manera activa a diferencia de Bob
- Alice utiliza **tls** para encriptar los mensajes, cuando se comunica con gitlab.com.
- Alice se pone a la escucha en el puerto **12345** mediante tcp, y se comunica mediante el **loopback** con Bob

Análisis de la ejecución conjunta de ambos procesos:

En esta sección vamos a analizar el comportamiento que tienen Alice y Bob cuando se ejecutan conjuntamente, no obstante, el análisis estático del Código no le vamos a repetir ya que obtendremos los mismos resultados.

Análisis de las llamadas al sistema:

Para ello vamos a utilizar strace y vamos a esperar a ver si la ejecución de ambos programas termina, los resultados de las llamas al sistema se han guardado en los archivos llamadas_bob_conjuntas.txt y llamadas_alice_conjuntas.txt. La primera diferencia que observamos es que ahora si ambos programas terminan su ejecución. Una vez obtenidos los

archivos vamos a probar a filtrar por las mismas palabras que cuando los analizamos individualmente para ver si algo ha variado.

- **llamadas_bob_conjuntas:** Al filtrar por **connect**, a diferencia de cuando de ejecutaba solo, ahora Bob solo realiza una sola vez la operación **connect** sobre la dirección 127.0.0.1:12345, ya que esta vez Alice se encontraba en ejecución y se pone a la escucha en esa dirección. También podemos observar que Bob al realizar la operación **read** sobre esta dirección 127.0.0.1:12345 obtiene información a diferencia de cuando se ejecutaba solo, cosa que era de esperar.
- **llamadas_alice_conjuntas:** La primera diferencia, la encontramos al filtrar por la palabra **write**, pues parece ser que Alice escribe sobre el descriptor de archivo 8 una serie de caracteres ilegibles, probablemente encriptados, y sobre el descriptor de archivo 5 siempre escribe un 0. Al filtrar por pipe, encontramos que crea una tubería con los descriptors de archivos 4 y 5, por lo tanto hemos decidido filtrar por **close** para ver si podemos encontrar información relativa a como el programa maneja las tuberías y descriptors de archivos, al filtrar obtenemos que realiza varias veces esta operación sobre los descriptors de archivos 3, 6 y 7. Al filtrar por read nos hemos encontrado que Alice lee a través del descriptor de archivo 7 un certificado, ya que encontramos lo siguiente **read(7, "-----BEGIN CERTIFICATE-----\nMIIF"..., 2022) = 2021**.

Análisis de la red:

A continuación, vamos a utilizar **wireshark** para observar la comunicación que mantiene Alice con el exterior y entre Alice y Bob, para ello primero vamos a analizar el tráfico de mi tarjeta de red wlo1 en **wireshark** para ver la comunicación con gitlab.com, posteriormente y sabiendo que Alice y Bob se comunican por el **loopback** analizaremos el tráfico de esa interfaz.

Análisis de la interfaz wlo1:

La comunicación que mantiene Alice parece ser la misma que en el caso en el que la ejecutamos de manera aislada, no obstante, tras finalizar la comunicación con gitlab.com, parece que Alice y Bob permanecen en ejecución bastante más tiempo, por lo que puede ser que Alice primero descargue unos datos de gitlab.com y posteriormente, realicen alguna operación entre ambos binarios. La información se encuentra en **info_alice** y el archivo se llama **captura_alice_conjunta_wlo1.pcapng**.

En total Alice obtiene de gitlab.com 1188188 bytes.

Análisis de la interfaz loopback:

Al ejecutarlo, observamos que Bob se conecta con la dirección 127.0.0.1:12345, una vez conectado, y hecho el **handshake**, se intercambian información que está totalmente encriptada por algún protocolo, además hemos observado que en esta ejecución han intercambiado un total de 2151301 bytes, no obstante, han tardado un total de 223 segundos, lo que resulta bastante tiempo para una transferencia de apenas uno MB. La información se encuentra en **loopback.pcapng**. Si observamos la imagen, podemos ver que la comunicación se realiza siempre mediante mensajes de 100 bytes.

Una vez realizado todos estos análisis, se ha pensado dos maneras de interceptar las comunicaciones de los binarios, la primera consistirá en intentar que la petición de DNS que se resuelva con la dirección 127.0.0.1 en lugar de la ip de gitlab.com, con el objetivo de que Alice piense que gitlab.com se encuentra en la máquina local y así intentar obtener la URL del recurso que Alice consulta en gitlab.com. La segunda, consistirá en intentar suplantar la identidad de bob, ya que la comunicación entre Bob y Alice parece ser una comunicación mediante TCP con los datos cifrados y así podríamos obtener el mensaje completo y luego intentar descriptarlo.

DNS Falso.

Para poder realizar esto, lo primero que necesitamos, es disponer de las herramientas de **netcat** y **openssl**, la primera nos servirá para crear un servidor en una dirección y puerto, en este caso **127.0.0.1:443** que es donde Alice realiza la petición DNS, la segunda nos servirá para crear una clave privada y un certificado que tendremos que auto firmar, para que cuando Alice comience la comunicación con la dirección anteriormente mencionada, este entregue dicha clave y certificado para cifrar las comunicaciones mediante tls. Para ello se han utilizado los siguientes comandos:

- Crear una clave privada: **"openssl genpkey -algorithm RSA -out private.key"** con este comando crearemos una clave privada, este se ha guardado en el archivo **private.key**.
- Crear el certificado auto firmado: **openssl req -x509 -new -key private.key -out certifie.crt** creando así el certificado, que se ha guardado en el archivo **certifie.crt**.
- **Modificar el archivo /etc/hosts:** en este archivo añadiremos una línea para que cuando Alice intente resolver el nombre de gitlab.com obtenga la ip 127.0.0.1, para ello en el archivo **/etc/hosts** añadiremos lo siguiente:

```
GNU nano 4.8 /etc/hosts
127.0.0.1    localhost
127.0.1.1    alonso-TUF-Gaming-FX505GD-FX505GD
127.0.0.1    gitlab.com
192.168.0.14 controller
192.168.0.21 compute
```

Una vez ya con todo esto, podemos comprobar que si realizamos nslookup gitlab.com obtenemos la ip de localhost, como se muestra a continuación:

```
alonso@alonso-TUF-Gaming-FX505GD-FX505GD:~$ nslookup gitlab.com
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
Name:   gitlab.com
Address: 127.0.0.1
```

Ya con todo esto ejecutamos el comando: `"nc -l -p 443 -k | openssl s_server -key private.key -cert certifie.crt"`, ya con el servidor funcionando vamos a probar a ejecutar a Alice y analizar el tráfico de red.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	45528 → 443 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1
2	0.000038077	127.0.0.1	127.0.0.1	TCP	76	443 → 45528 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495
3	0.000070136	127.0.0.1	127.0.0.1	TCP	68	45528 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=394034662
4	0.000093212	127.0.0.1	127.0.0.1	TLSv1	348	Client Hello
5	0.00015775	127.0.0.1	127.0.0.1	TCP	68	443 → 45528 [ACK] Seq=1 Ack=281 Win=65280 Len=0 TSval=3940346

No obstante, las comunicaciones siguen cifradas y no hemos conseguido descifrarlas, por lo que tras esta prueba no hemos obtenido ningún resultado.

Suplantar a Bob.

Como mencionamos anteriormente cuando analizamos el tráfico de **loopback**, Bob intentará conectarse al puerto 12345 e ip 127.0.0.1 mediante TCP, por lo que podríamos probar a crearnos nuestro propio socket e intentar conectarnos a esa dirección, además, Bob cada vez que Alice envía 100 bytes, este le responde con un OK cosa que también realizará nuestro programa impostor, asique para ello vamos a realizar un script de python para realizar estas tareas, que se muestra a continuación.

```
1  #!/usr/bin/env python3
2
3  import socket
4  import base64
5
6  ip = "127.0.0.1"
7  port = 12345
8
9  client_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 client_sock.connect((ip, port))
11 bytes_descriptado = b''
12 data = b''
13 read = b''
14
15 while True:
16     read = client_sock.recv(1024)
17     data += read
18     if not read:
19         break
20     read = b''
21     client_sock.send(b'OK')
```

El script es bastante sencillo, ya que simplemente es un bucle while infinito el cual primero recibe datos de la dirección 127.0.0.1:12345 (que se corresponde con Alice) los concatena con los anteriores, y en el caso de no haber recibido nada, sale del bucle infinito, tras probar el script y analizar el tráfico de red ejecutando Alice y FakeBob.py obtenemos los mismos resultados que si de Bob se tratase, incluso a pesar de realizar un **recv** de 1024 solo nos llegan tramas TCP de 100 bytes por lo que podemos decir que Alice envía los datos de 100 en 100 bytes. La captura de wireshark se encuentra en el archivo **FakeBob.pcapng**.

Suplantando a Bob hemos conseguido interceptar la comunicación, no obstante la información sigue estando cifrada, para intentar descifrarla hemos consultado algunas páginas de criptografía en internet con el objetivo de encontrar más información sobre la posible forma de cifrar el mensaje, en la siguiente página web : <https://brianur.info/cifrado-caesar/> hemos encontrado información interesante, y es que al insertar parte del mensaje cifrado, obtenemos algunas

palabras legibles al utilizar el cifrado cesar y una semilla $N = 8$, como se muestra en la siguiente imagen.

Mensaje a cifrar / descifrar :

\xf7\x3
\xc7Xzqumzi(xiz|m(lmt(qvomvqw{w(pqlitow(lwv(Y}qrw|m(lm(ti(Uivkpi

Semilla : | |

Resultado del criptograma

\PX7\PU3
\PU7PRIMERA(PAR|E(DEL(INGENIO{O(HIDALGO(DON(Q}IJO|E(DE(LA(MANCHA
\P15\P12\P15\P12\P15\P12\P15\P12\P15\P12\P15\P12CAP
\PUT\PT5|}LO(PRIMERO6(Q}E(|R

Si observamos detenidamente, podemos darnos cuenta de que por ejemplo la letra i en el mensaje encriptado corresponde con la letra a en el mensaje original, y así para el resto de las letras, por lo tanto, podríamos probar a coger el mensaje que Alice envía a Bob y aplicarle este algoritmo a todos los bytes para intentar obtener el mensaje. Para ello modificamos el archivo FakeBob.py incluyendo las siguientes líneas al final del bucle while:

```
bytes_desencriptado = bytes((byte - 8) for byte in data)
print(bytes_desencriptado.decode())
client_sock.close()
```

Una vez hecho esto, vamos a probarlo, para ello ejecutaremos Alice y el script FakeBob.py y redirigiremos la salida estándar a un archivo llamado **MSG_desencriptado.txt**.

Tras finalizar la ejecución de ambos programas, obtenemos el mensaje totalmente desencriptado, y podemos observar que lo que Alice le enviaba por el **loopback** a Bob era ni más ni menos que un capítulo del libro del Quijote.

Conclusiones.

En resumen, podemos decir que Alice obtendrá de algún repositorio de gitlab el capítulo del Quijote, realizando una petición al puerto 443, y cifrando la comunicación por tls, además Alice creará un socket TCP y se pondrá a la escucha en el puerto 12345 y en la ip 127.0.0.1, Bob se conectará y empezarán una comunicación para transmitirse el texto cifrado, finalmente al terminar la transmisión ambos programas finalizan.

